Prefetching the Means for Document Transfer: A New Approach for Reducing Web Latency

Edith Cohen
AT&T Labs–Research
180 Park Avenue, Florham Park, NJ 07932

Haim Kaplan
Tel Aviv University
Tel Aviv 69978, Israel

Abstract—

User-perceived latency is recognized as the central performance problem in the Web. We systematically measure factors contributing to this latency, across several locations. Our study reveals that DNS query times, TCP connection establishment, and start-of-session delays at HTTP servers, more so than transmission time, are major causes of long waits. Wait due to these factors also afflicts high-bandwidth users and has detrimental effect on perceived performance.

We propose simple techniques that address these factors: (i) *pre-resolving* **host-names (pre-performing DNS lookup), (ii)** *pre-connecting* **(prefetching TCP connections prior to issuance of HTTP request), and (iii)** *pre-warming* **(sending a "dummy" HTTP `HEAD` request to Web servers). Trace-based simulations demonstrate a potential to reduce perceived latency dramatically. Our techniques surpass document prefetching in performance improvement per bandwidth used and can be used with non-prefetchable URLs.**

Deployment of these techniques at Web browsers or proxies does not require protocol modifications or the cooperation of other entities. Applicable servers can be identified, for example, by analyzing hyperlinks. Bandwidth overhead is minimal, and so is processing overhead at the user's browser. We propose scalable deployment solutions to control the potential overhead to proxies and particularly to Web servers.

I. INTRODUCTION

The central performance problem of the Internet today is user-perceived latency, that is, the period from the time a user issues a request for a document till the time a response is received. A key realization is that latency is often NOT dominated by document transmission time, but rather by the setup process that precedes it. This realization was a central motivation for HTTP/1.1, which addressed connection-establishment time on subsequent HTTP requests [22] and for proposals such as prefix-caching of multi-media streams [27]. Even users enjoying persistent HTTP and high-bandwidth connectivity, however, are still frequently afflicted with annoying long setup waits. We propose natural techniques to address dominating latency causes that precede document transfer.

Background. Communication between Web clients and servers uses the HyperText Transfer Protocol (HTTP), which in turn utilizes Transmission Control Protocol (TCP) as the *de facto* underlying reliable transport protocol. A TCP connection needs to be established and acknowledged prior to transporting HTTP messages. To facilitate connection-establishment, the Web server's host-name representation is translated to a numeric Internet Protocol (IP) address. This translation is done by querying a DNS (Domain Name System) server that may consult a hierarchy of DNS servers. Determining factors of the user perceived latency are name-to-address resolution, TCP connection-establishment time, HTTP request-response time, server processing, and finally, transmission time.

Practice with HTTP/1.0 was to use a separate TCP connection for each HTTP request and response [6]. Hence, incurring connection-establishment and slow-start [17][1] latencies on each request [15]. Persistent connections [22] address that by re-using a single long-lived TCP connection for multiple HTTP requests. Persistent connections became a default with HTTP/1.1 [14], which gets increasingly deployed. Deployment of HTTP/1.1 reduces the latency incurred in *subsequent* requests to a server utilizing an existing connection, but longer perceived latency is still incurred when a request necessitates establishment of a new connection.

Document caching and prefetching are well studied techniques for latency reduction. Caching documents at browsers and proxies is an effective way to reduce latency on requests made to *cache-able previously-accessed* documents. Studies suggest that due to the presence of cookies, CGI scripts, and limited locality of reference, caching is applicable to only about 30%-50% of requests [20], [11], [13]. Furthermore, caching is also limited by copyright issues. To avoid serving stale content, cached resources are often validated by contacting the server (e.g., through an `If-Modified-Since GET` request.) Hence, caching eliminates transmission time, but often, and unless pre-validation is used [19], [9], still incurs considerable latency. Document prefetching reduces latency by predicting requests and initiating document transfer prior to an actual request. The effectiveness of document prefetching is limited by accuracy of predictions and the availability of lead time and bandwidth [20], [9], [10]. The use of document prefetching is controversial due to its extensive overhead on network bandwidth.

Our contribution. We systematically measure several latency factors and study their sensitivity to *reference locality*. We propose techniques that address significant latency factors by *prefetching the means* to document transfer (rather than the document itself). Finally, we conduct performance evaluation of our techniques by replaying actual users request sequences. Next we overview conclusions from our measurements, our proposed techniques, and their performance evaluation.

`1) Measurements.` We used a list of about 13 thousand Web servers extracted from a proxy log. The measurements were conducted from several locations in order to ensure they are not skewed by unrepresentative local phenomena. Below we summarize our findings.

- *DNS lookups.* DNS lookup time (name-to-address translation) exceeded 3 seconds for over 10% of servers. Lookup time

[1] *TCP slow start* refers to the several round trip times (RTTs) until TCP finds a "correct" transmission speed.

is highly dependent on reference locality *to the server* due to caching of query results at name servers.

• *Cold- and warm-server state.* We observed that request-response times of start-of-session HTTP requests are on average significantly longer than of subsequent requests (even when each request utilizes a separate TCP connections). Presumed start-of-session HTTP request-response time exceeded 1 second for over 10% of servers and exceeded 4 seconds for over 6% of servers. These fractions dropped by more than half for subsequent requests.

• *Connection establishment time.* In agreement with many previous studies (e.g. [22], [13]), we observed that TCP connection establishment time is significant relative to HTTP request-response times.

• *Cold and warm route states.* As we explain later in the paper the first IP datagram traversing a path to a destination (when the route is cold) is more likely to travel for a longer time period than subsequent datagrams (when the route is warm). This effect is visible on consecutive TCP connection-establishments times, but our study concludes that is not as significant a contributor to long latency as other factors.

Furthermore, our study shows that DNS lookup time, connection-establishment time, and HTTP request-response time all exhibit heavy-tail behavior (i.e. the average is considerably larger than the median).

`2) Solutions.` We propose the following techniques to address the three latency factors described above.

• *Pre-resolving.* Browser or proxy performs DNS lookup before a request to the server is issued thereby eliminating DNS query time from user-perceived latency.

• *Pre-connecting.* Browser or proxy establishes a TCP connection to a server prior to the user's request. Pre-connecting addresses connection-establishment time on the first request utilizing a persistent connection.

• *Pre-warming.* Browser or proxy sends a "dummy" HTTP `HEAD`[2] request to the server prior to the actual request. Pre-warming addresses start-of-session latency at the server.

We refer to the three techniques combined as *pre-transfer prefetching*.

Pre-connecting complements persistent HTTP and caching of TCP connections [8], [13]: It addresses wait due to connection-establishment incurred on the *first request* utilizing a connection. Pre-transfer prefetching techniques also complement caching of documents since 1) they are more beneficial for requests to servers which were not recently accessed and 2) their effectiveness does not depend on the URL being a cache-able document. They complement document prefetching by providing a range of overhead/benefit tradeoffs and utilizing less bandwidth. Similar to document prefetching, our pre-transfer techniques require a scheme to predict servers that are likely to be accessed.

`3) Performance.` We evaluated the effectiveness of these techniques on two classes of requests across two locations.

The first class of requests was search-engines referrals, namely follow-up requests on results returned by search-engines or Web-portals (directories). We chose to focus on these requests since search-engines and Web-portals sites typically invest considerable effort to provide low-latency high-quality service. Performance on follow-up requests is crucial in the overall perception of performance, however, we observed that follow-up requests are subjected to considerably longer latencies than average HTTP requests. Perceived latency on AltaVista [1] referrals exceeded 1 second for 22% of requests and exceeded 4 seconds for 12% of requests. With pre-resolving, pre-connecting, and pre-warming (with pre-connecting) applied, latency exceeded 1 second only for 10%, 6%, and 4% of requests, respectively. Latency exceeded 4 seconds for only 4%, 3%, and 2% of requests, respectively, yielding a dramatic decrease in perceived latency.

The second class of requests was considerably wider, containing all requests that were not preceded by a request to the server in the last 60 seconds. Latency exceeded 1 second on 14% of requests and exceeded 4 seconds on 7% of requests. With pre-resolving, pre-connecting, and pre-warming, latency exceeded 1 second only on 9%, 5%, and 3% of requests, respectively. Latency exceeded 4 seconds on 4%, 2.5%, and 1% of requests, respectively. This demonstrated dramatic potential improvement in performance.

Outline. The remainder of this paper consists of six sections. Section II describes our data and how we performed latency measurements. Section III describes our measurements of the different latency factors. Section IV discusses pre-resolving, pre-connecting, and pre-warming and their deployment. Section V lists possible overheads and suggests ways to address them. Section VI presents the performance evaluation. We conclude in Section VII and propose directions for further research.

II. DATA

Our user activity data was a log of the AT&T Research proxy server which is described in detail in [23]. The log provided, for each HTTP request, the time of the request, the user's (hashed) IP address, the requested URL and Web server, and the referring URL. The log contained 1.1 million requests issued between the 8^{th} and the 25^{th} of November 1996 by 463 different users (IP addresses). Requests were issued to 17 thousand different Web servers, and for 521 thousand different URLs.

We measured different latency components by a simple "Web browser" program utilizing a BSD-based socket interface. Our program replayed logged requests and timed latencies. We conducted measurements from 3 locations: AT&T Shannon Lab in Florham Park, NJ, Stanford University in Stanford, CA, and Tel Aviv University in Israel. This methodology, rather than packet trace analysis, allowed us to compare latencies associated with the same request sequence across locations, run various "what if" scenarios to evaluate our techniques, measure factors not easily available from a packet trace, be up-to-date (the original log is over 2 years old), and systematically study the dependence of various latency factors on reference locality.

As mentioned in Section I it was important to identify requests that had occurred when a user was browsing a search engine or Web portal query response page. We did that by using the *referer field* that is logged with each request. As representative sites we chose AltaVista™ [1] and Yahoo!© [30]. There

[2] `HEAD` requests are identical to `GET` except that the response contains only the header of a document.

were 2942 Altavista referral requests (for which we obtained a valid HTTP response) to 2151 different servers, and 2452 Yahoo! referrals to 1397 servers.

III. Study of latency factors

We measured the different latency factors and study the effect of reference locality on them. A measurement was taken for each server in a group and results were aggregated. We aggregated the measurements across the complete set of servers ocurring in the AT&T research proxy log described in Section II, and across the subsets of servers accessed as a result of a referral of a particular search engine or Web portal. The dominating latency factors were the same for both sets of servers so we show results for servers referred by AltaVista unless explicitly mentioning otherwise. Measurements taken at Stanford and at AT&T labs were similar so we show only the ones taken at AT&T labs. Note that the actual request sequence was not used here (we did use it for performance evaluation in Section VI).

We measured the followings: DNS lookup time, TCP connection-establishment time, and HTTP request-response time. We summarize our findings below.

Format of the figures. Our figures typically show a distribution of latency measurements. We show the fraction of measurements *exceeding* x, which corresponds to $1 - F(x)$, where $F(x)$ is the cumulative distribution function of the measurements. We found this less-conventional presentation natural for our study, since our focus is on the (relatively small) fraction of requests with long latencies, rather than the median or majority behavior.

A. DNS lookup time

When a browser or proxy has to map a domain name to an IP address it sends a DNS query to a local name-server. The name-server may have the answer in its cache and when it does not, it communicates with other name-servers to obtain the address. A name-server caches an address for TTL (Time To Live) seconds. The TTL parameter for a particular host is determined by its system administrators, and most TTLs are 24 hours or less [7]. Some browsers cache results of DNS queries for a time period independent of their TTL (default value used at Netscape's browser Mozilla is 15 minutes [24], and some browsers simply deploy a limited-size name-to-address cache).

Figure 1 shows DNS lookup times, from 3 different locations, for servers resolved successfully. DNS queries issued from Tel-Aviv University (TAU) reflect longer network RTT to the US (typically 500ms). DNS query times across all 3 locations exhibit a similar pattern of "flat areas," where very few query times are between 1 and 3 seconds or between 8 and 16 seconds. This is due to commonly-used timeouts-and-retransmit values. Also evident are long query times, exceeding 4 seconds for about 10% of servers. Query times show a heavy-tail behavior, where some successfully-completed queries lasted minutes.

We also measured the lasting affect of caching at DNS servers. Our program issued DNS queries with varying time intervals since previously-issued identical queries. Our measurements show that query times tend to increase as the time from a previous identical query increases. This is due to caching, at the local name-server and other name-servers it communicates with, of query results or partial results (that assist in speeding up the resolution process.) The gradual attenuation of the effect of a previous query over time reflects this distributed nature of the resolution process. The effect levels off completely at around 3 days.

B. Connection-establishment time

TCP connection-establishment time incorporates the RTT over the network (of a single SYN segment), retransmissions of SYN segments due to timeouts (e.g., server had a full listen queue and discarded the SYN), and any other processing needed to facilitate the connection.

Each router along the way does some work for every IP datagram which belongs to the particular TCP connection and travels through it. First, the routing algorithm (e.g. OSPF) is consulted to determine the IP address of the next hop, based on the destination. Second, the IP address of the next hop is translated to a wire-level address (e.g. Address Resolution Protocol (ARP) is used to translate IP-address to MAC address). These operations may require communication with other routers. The results of this processing may be cached by a router for a period of time, enabling faster routing of subsequent IP datagrams to the same destination. Therefore, it is conceivable to expect the first RTT of the datagrams corresponding to the SYN segments will be longer than subsequent ones. This is the so called *cold route effect*.

Figure 2 shows connection-establishment times for 4 consecutive connection-establishments named s1, s2, s3, and s4, respectively. The effect of route warming is visible but small for short time frames, where s1 is consistently above s2,s3, and s4. This dissipates for longer RTTs when the effect of route warming is small compared to total establishment time. Figure 2 also shows the median of s2, s3, and s4 which is consistently below either of them demonstrating the heavy-tail behavior of connection-establishment time. The "flat" areas between 4 and 6 seconds and over 8 seconds in Figure 2 is likely to be due to timeouts and SYN-retransmissions performed with connection-establishment. BSD uses about 6 seconds for the first timeout and 24 seconds for the second [28].

Our measurements (omitted due to space limitation) also show that connection-establishment time varies significantly with the time-of-day. In fact, the route-warming effect is small relative to the variability in connection establishment time due to the time-of-day. To summarize, although we observed that the cold route effect is weak, the overall connection establishment time is significant especially in busy hours. Typical HTTP response is short and may fit into a single HTTP segment. In such cases there would be only one round trip following the connection-establishment round trip and the connection-establishment time will be approximately half the time of the whole transaction.

C. Cold- and warm-server request-response times

The following experiment gauges transmission time and HTTP request-response time. We say that a Web server is in a *warm state* with respect to a particular user when it recently got an HTTP request from that user, and it is in a *cold state*

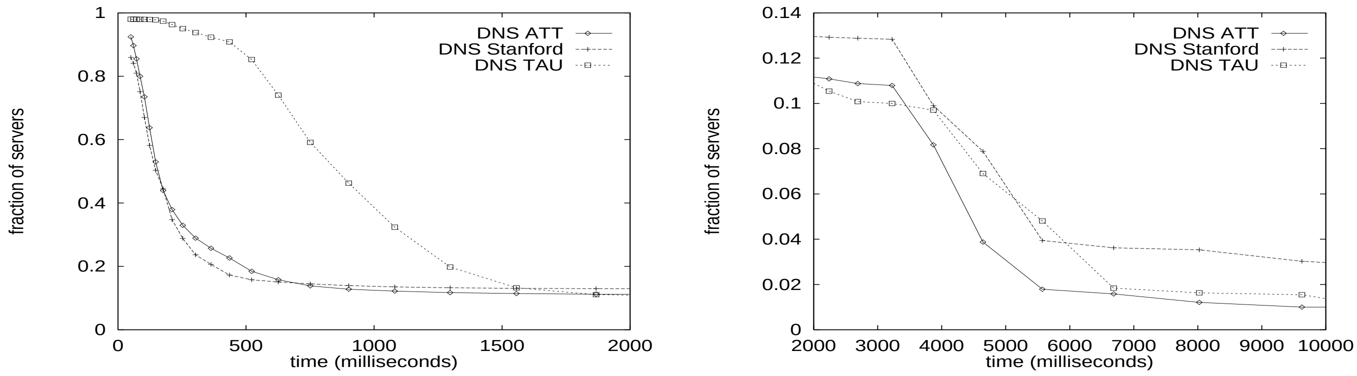

Fig. 1. DNS query time.

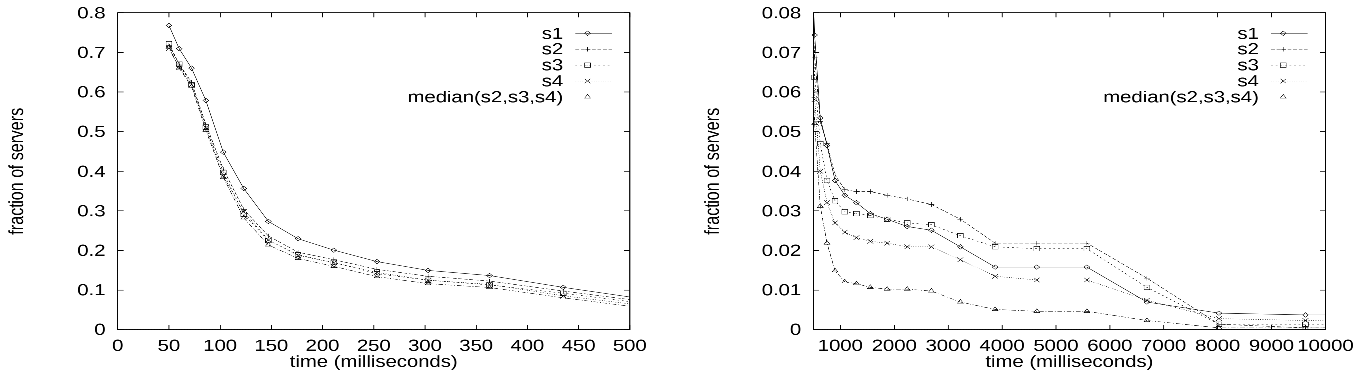

Fig. 2. Connection-establishment times for 4 consecutive connection-establishments of TCP connections.

otherwise. We identify a considerable gap in HTTP request-response time of warm and cold servers. We timed 4 consecutive TCP connection-establishments (SYN1–SYN4), 3 consecutive HTTP `HEAD` requests (HEAD1–HEAD3) followed by 3 consecutive `GET` requests (GET1–GET3). Each HTTP request utilized a separate TCP connection. The "difference" between the time of `HEAD` and `GET` requests is indicative of file transmission time. A `GET` request may require transmission of a larger segment or more than one segment to transfer the whole file from the server back to the client. We note that our measurements were performed in high-bandwidth environments, resulting in short observed transmission times. Transmission time is more significant for low-bandwidth modem users [23]. We requested "home-pages" of servers using HTTP/1.0 for two reasons. First, requests for home-pages put all servers on a more equal footings, and eliminate variance in RTTs to different servers due to server processing time and transmission time. Second, a measurable fraction of the URLs in our log were no longer valid.

Figure 3 shows the respective times. As expected, behavior within each of the groups SYN2–SYN4, HEAD2–HEAD3, and GET1–GET3 was very similar, and only one representative from each group is included in the figure. We describe the measured items from shortest to longest latency in increasing order (from the low curve and up in Figure 3).

TCP connection-establishment time on a warm route (SYN4) was on average shortest. Next is TCP connection-establishment on a possibly cold route (SYN1). (Notice that we cannot guarantee a cold-route or a cold-server state since we performed our measurements on active systems.) HTTP `HEAD` request-response (HEAD3) on a warm server are next, indicating that HTTP application processing is a more significant cause of latency than a cold route. The gap between HEAD3 and SYN4 is indicative of time spent by the HTTP application of the server when in a warm state. It may also reflect the different timeouts TCP uses for SYN segments and later segments. Above the HEAD3 curve is the HTTP `GET` request-response time (GET1), which "adds" transmission time to HEAD3. On short request-response times (200ms–600ms), warm-server processing and transmission time contribute visibly to the total latency. Combined, they double the number of request-response times above 200ms–600ms. The measurements for SYN1, SYN4, HEAD3, and GET1 become clustered together for time intervals exceeding 1 second, as the network RTT increasingly dominates HTTP request-response time. The time of TCP connection-establishment often exceeds the HTTP request-response time, possibly due to longer timeouts for SYN segments (TCP set the retransmission timeout for later segments based upon the measured RTT) and queues at loaded servers.

Significantly above the other curves and exhibiting longer HTTP request-response times is the HEAD1 curve, which corresponds to time spent by the HTTP application of (possibly) cold servers. The gap between HEAD1 and HEAD4 corresponds to the difference between cold and warm-server states. The plot indicates that a significant fraction of Web servers exhibit significant cold-state slow-downs. The gap widens as the time increases, revealing the cold-server state as a major cause of long latency. The HTTP `HEAD` request-response time exceeded 2 seconds for about 5% of servers when cold and only for about

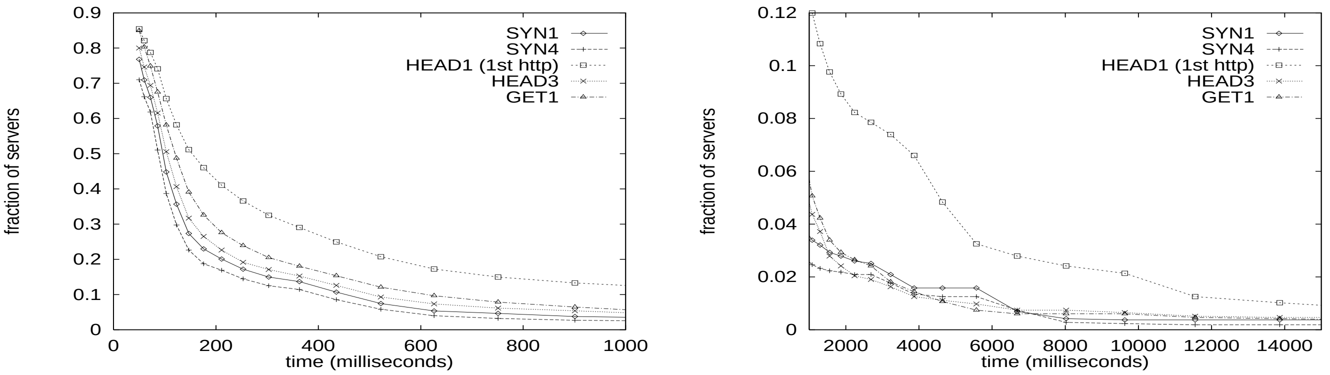

Fig. 3. Measurements of consecutive TCP connection-establishments, 3 consecutive `HEAD` requests followed by 3 consecutive `GET` requests.

2% of servers when warm. Generally, the number of HEAD1 (cold server) measurements exceeding any time between 600ms and 16 seconds is double or triple the corresponding number for HEAD3 (warm server). A study of the lasting effect of warm servers indicates that servers seem to remain warm for about 15 minutes with no evident gradual change.

A major contributer to this phenomenon seems to be reverse-DNS query issued by some Web servers for every request. Due to caching of DNS query results, subsequent requests from the same client are processed faster. Server may use the client's domain name in order to implement domain-name-based access control (supported by commonly-used HTTP server software like Apache [2] and NCSA [25]), for logging purposes, or for targetted content distribution [21].

To summarize, we observe that DNS query time and a cold-server state are major causes of long latencies. In agreement with previous studies, we observe that TCP connection-establishment delay is significant with respect to HTTP request-response time.

IV. Combating Latency

We propose three techniques to address the major latency factors identified in Section III. In essence, our techniques perform the setup work associated with an HTTP request-response prior to the time the user issues the request. All our proposed techniques are applicable to *servers* rather than *documents*, and require minimal bandwidth. The general paradigm is to apply each technique to a set of servers with above-threshold likelihood to be accessed within the effective period of the technique. The precise method and threshold to use in order to identify such servers can vary for different techniques. Our techniques aim at reducing setup latency and are effective for requests where the previous access to the server was not very recent. As such, they are not applicable when there are no setup costs, that is, when an existing persistent connection is used. We present each of three techniques below.

Pre-resolving. Perform the DNS query prior to the user request and eliminate the long query time from the user perceived latency. For example we can pre-resolve all server names that appear in AltaVista response page, or more generally a subset of the servers represented in hyperlinks on the currently-browsed document. According to our measurements in Section III, the effects of pre-resolving may last from minutes to days. One can also program browsers to identify for each individual user a list of servers that the user is likely to access in a browsing session. The browser then can pre-resolve the whole list each time the user starts a browsing session.

We mention that another way to reduce DNS latencies is to improve the caching mechanism of DNS itself. In [7] we propose and evaluate specific techniques for achieving that.

Pre-connecting. *Pre-connecting* is the establishment of a TCP connection prior to a user request. As mentioned in Section III, when the requested document is of small size, it is often the case that only two round trips are required to fetch the document: one for connection-establishment and the other for the document transfer itself. Pre-connecting eliminates one of these RTTs from the user latency. When an IP address is not yet available pre-connecting requires pre-resolving. Pre-connecting is effective when performed within a time period where the server is likely to keep an idle connection, typically tens of seconds or minutes. The Apache server [2] uses a configurable timeout with default setting of 15 seconds.

Pre-warming. *Pre-warming* a server amounts to issuing a "dummy" HTTP request, prior to an actual request. The "dummy" request could be an HTTP `HEAD` request, which requests only a header of the document and therefore uses minimal bandwidth. When an actual request is issued, the server is in a warm state (e.g., results of a reverse-DNS query are cached at the server). We observed that pre-connecting alone does not result in a warm server (perhaps also because access control is performed by the HTTP application). Pre-warming seems to remain effective for about 15 minutes. It requires connection-establishment and also DNS lookup when IP-address is not available.

V. Scalable Deployment

Our pre-connecting techniques have minimal bandwidth overhead but they do impose other overheads: Pre-resolving increases the number of DNS queries, pre-connecting increases the number of connection-establishments as well as the number of idle TCP connections, and pre-warming results in additional work for the HTTP application. Whereas the benefit of these techniques is to the end users the cost incurred mainly in the network and in the servers. This discrepancy between location of cost and benefit is not as strongly expressed even for Web document prefetching. The latter is often self-limiting by bandwidth in local links. Direct experimental evaluation of these overheads is complicated. Therefore, we provide a qualitative discussion and suggest ways to cope with potential problems.

We start with an overview of the overhead expected at various network entities. We go from entities that see the least overhead to entities with a larger overhead.

Browsers. Pre-resolving, pre-connecting, and pre-warming can be fairly easily incorporated in the browser. Web browsers routinely open multiple connections even to a single host, and workstations can support dozens of simultaneous TCP connections. In the implementation, one may need to be careful and increase the size of the connections cache of the browser. Mozilla [24], for example, seems to have a limit of 15 connections which are managed LRU-style when idle. Microsoft Explorer, on the other hand, uses fixed-timeout expiration and no hard limit on connection-cache size [29]. The second implementation is more appropriate for pre-connecting browsers.

Proxies. Often *proxy servers* bridge users and content servers. In such setup, TCP connections to the servers and HTTP requests and responses are managed by the proxy. Therefore, pre-resolve, pre-connect, and pre-warm requests should either be conveyed to the proxy or be initiated by the proxy as part of the service. As a result, proxy servers may need to issue and handle increased numbers of connections.

DNS servers. DNS servers may see increased number of queries due to "unused" pre-resolves. This can particularly affect the root DNS servers.

Routers. Routers are not subject to a significant *bandwidth* increase, but there is an increased number of *different IP packet flows* being routed. Routers may cache next-hop address for destinations in recently-seen packets. Hence, the change in the balance of total bandwidth and number of different destinations may affect performance.

Web servers. Web servers will see increased numbers of TCP connection establishments and idle TCP connections. Prewarms also inflict additional work on the HTTP session.

A. Coping with the overheads

There are few immediate steps proxies, routers, DNS servers, and Web servers can take to reduce the extra overhead. Proxies can multiplex many users' traffic on the same TCP-connection to the server and hence, pre-connected connections can be shared. Similarly, the benefits of a single pre-resolve or prewarm can be applicable to several users. Sharing reduces overhead at the proxy itself, servers, and the network. Multiplexing users over TCP connections by proxies has been proposed before in the context of persistent HTTP [18].

Routers may tune themselves for the change in the balance of total bandwidth and number of different destinations. They, for example, can cache route related parameters only after *at least few packets are routed* rather than just the first one. This strategy would reduce the use of valuable router cache space on pre-connects or pre-warms traffic (while minimally affecting the benefit of these techniques, according to measurements in Section III.) This heuristic can be implemented using a simple randomized strategy: "flip a biased coin" for each datagram and cache the next hop to its destination if the outcome is "heads". Such an implementation will not require maintaining counters associated with different destinations.

The DNS system incorporates mechanisms for having more than a single name-server authoritative for each zone. The original intention was mainly to achieve high degree of fault tolerance. Nevertheless, the same mechanism is already used for load-sharing on heavily-queried zones and can be used for handling increased load due to pre-resolving. Other techniques for improving the DNS caching mechanism that we suggest in [7] may also help.

The highest overhead of pre-transfer techniques would probably be incurred at Web servers or reverse-proxies, which would have to handle a larger number of TCP connections and more HTTP pre-warming requests. To put this cost in perspective, however, we note that pre-warm/pre-connect cost is cheaper to the server than full-fledged document prefetching. Therefore, overhead is actually reduced if we use the techniques only as a network- and server-friendly *substitute* for document prefetching. We expect pre-connects, however, to be more abundant due to lower overhead. Even without pre-transfer techniques, however, current use of TCP under HTTP involves multiple connection establishments and idle TCP connections: HTTP/1.1 promotes connection caching and keeps a TCP connection idle and open for a period of time following a request. Furthermore, under both HTTP/1.0 and HTTP/1.1, browsers open multiple TCP connection to a single server. Careful studies of the effect on Web servers of multitudes of TCP connections-establishments and idle connections were performed by [3], [5], [16]. These studied indeed concluded that under current architectures/OS setups, this can have a detrimental effect on servers' throughput. On the positive side, these limitations do not seem to be all inherent in the protocols, but are more due to implementations. Furthermore, solutions would address not only overhead due to pre-transfer techniques but also current practices. For example, connections backlog can often be avoided with tuning of kernel parameters. Servers can support increased number of connections by enhancing the operating system [4] or adaptive allocation of buffer sizes [26]. Another approach to relieving the load is to simply increase the number of machines. This is consistent with current trends of Web content hosts, recognizing the fact that adding machines is relatively cheap. Conceivably, however, the overhead of pre-connecting may become particularly significant for popular servers with many hyperlinks to them. Therefore, we propose the following guidelines for deployment of our techniques.

B. Deployment guidelines

Many aspects of the Web, including TCP, rely on good "social behavior." Hence, it is as reasonable to expect implementations of pre-transfer techniques as part of browsers or proxies to follow guidelines. One such guideline that we already mentioned is sharing of connections and pre-warms across their clients. Another guideline is for browsers and proxies to terminate prefetched connections as soon as access likelihood drops (e.g., when the user switches context). This limits the number of idle connections servers have to maintain. Also, connections that are terminated by client avoid TIME_WAIT state at the server. (The closing party gets into a TIME_WAIT state to avoid delayed packets associated with the closed connection to be accepted by a later connection between the same hosts.) Accumulation of connections in TIME_WAIT state at the server may severely degrade its performance [12]. Curiously, current practice by popular browsers is not to terminate connections when context is switched (Microsoft IE uses fixed-timeout and Navigator uses fixed-size connection cache which terminates a connection only when a new request arrives [29]).

The different pre-transfer techniques have varied effectiveness periods: pre-resolving lasts for hours or days, pre-warming is effective for about 15 minutes, and pre-connecting is effective for tens of seconds. The techniques vary in overhead as well: Pre-resolving incurs least overhead and pre-warming incurs the most overhead. Good algorithms need to take into account the cost-benefit trade-offs of the different techniques. in order to achieve good trade-offs of *recall* (fraction of requests benefiting from a pre-resolve/pre-connect/pre-warm) and *precision* (fraction of pre-resolves/pre-connects/pre-warms that end up being used).

A prefetching algorithm requires a *prediction scheme* to predict which servers are likely to be accessed. Prediction can be based on analyzing hyperlinks, extracting implication rules from browser data across users, or tracking same-user browsing patterns. In our experiments we assumed hyperlink-based prediction in the context of search-engines follow-up requests. The slew of prediction schemes proposed in the context of document prefetching may be applied with our techniques.

Another important component is a *pricing scheme* to evaluate overheads. Pre-connects and pre-warms to unloaded Web servers incur minimal cost, but pre-connects to loaded servers cause performance degradation. Overheads in the context of document prefetching are mostly measured by the relative increase in bandwidth and sometimes also by local storage space. These metrics are negligible in our contexts. A possible pricing scheme for our techniques is associating a fixed overhead cost with each pre-warm and pre-resolve and a fixed overhead cost plus a quantity proportional to the connection open time for pre-connects.

Once we have a prediction scheme and a pricing scheme we can use them by applying a pre-transfer technique if the ratio between (estimated) latency-cost multiplied by the (estimated) likelihood of access and the (estimated) overhead cost exceeds a certain threshold. To control cumulative overhead to particular servers, it is important to pre-connect to such servers only if likelihood of access exceeds a threshold. For example, a user should pre-connect to `www.altavista.com` only if there is at least say, 10% chance that it will indeed send `www.altavista.com` a real request soon. Such rule would guarantee that the number of pre-connects issued to `www.altavista.com` is within bounded ratio of the number of connections that end up being used.

C. Protocol enhancements

Wide-spread pre-connecting and pre-warming may justify protocol enhancements which increase efficiency and control the impact on popular sites. One idea is to use the DNS to communicate additional information from server sites to clients. This can be performed, for example, via new kinds of Resource Records (The DNS stores all the information in few kinds of Resource Records) or by placing additional information in the existing address records. Using such a mechanism a server may request that no pre-connects, pre-warms, Web crawl, or offline cache validation requests are made to the site (this can also be limited to specific times-of-day).

Another proposal applicable to pre-warming is to add a new `access` request to HTTP, that would cause the server to perform start-of-session processing for the client. This would allow servers to distinguish pre-warming requests from actual requests and thereby giving them low priority in times of congestion.

VI. Performance Evaluation

We compared perceived-latency with and without applications of each of our techniques. We used the request sequence in the AT&T proxy log (see Section II), and subsets of it consisting of follow-up requests after queries to search engines or Web portals. Our motivation for considering these subsets is described in detail in Section I.

A. Methodology

Instead of a complicated "real-time" replay of the log we accounted for reference locality in the following way. For each server represented in the log, we timed the different latency factors associated with an HTTP request. Multiple measurements were taken for each server with varying time-intervals following a previous identical request made to the server. Let r be a logged request. We define $\Delta(r)$ to be the difference between the logged time of r and the logged time of a previous request to the same server as the one accessed by r. Whether we consider the previous request to the server from the same user or the previous request to the server from any user depends on the scenario we simulate as we discuss below. The latency associated with a logged request r is the latency measured approximately $\Delta(r)$ seconds after a previous request to the server. All measurements were taken in a relatively short time interval for all servers. We nullified the DNS component of the latency for requests with $\Delta(r)$ less than 15 minutes. This is to account for caching of DNS query responses at browsers and is consistent with the default value used in Mozilla [24].

Our measurements were performed on an active system and hence, the values reflect other users' activities over which we had no control. We considered two scenarios. The first scenario assumed that a single user (at a time) from our log is active on top of the current environment. To simulate that we used the previous logged request to the server *made by the same user* to compute $\Delta(r)$. In the second scenario all logged users are active on top of the current environment. To simulate that, we used the most recent previous request to the server *made by any user* to compute $\Delta(r)$.

The time available for prefetching, which we call *lead time*, is the time since we have all the information to identify the server that we want to pre-connect to, until the time the actual request to the server may arrive. Reduction in perceived-latency is bounded by the lead time available for prefetching. Small lead time may prevent us from completing the pre-transfer prefetching process before the actual request. It is also often desirable for lead time not to be too large. In a case when lead-time is large we may pre-connect too early so by the time an actual request comes the connection is terminated. We estimated the lead time of a request referred by a search engine as the time from the previous request made by the same user to the particular search engine[3].

For the group of search engine referrals we evaluated performance in two ways: 1) taking lead time into account and disallowing latency reduction that exceeds the lead time. 2) assuming that the lead time always exceeds the perceived-latency reduction. We shall refer to this assumption as the *sufficient lead time* assumption. When evaluating performance for search engine referrals we assumed that requests with $\Delta(r)$ at most 30 seconds do not require connection-establishment. Such requests are mostly issued by the same user after the same query (maybe using the "back" button of the browser in-between) and likely to utilize a persistent connection.

When evaluating performance on the complete log we used only requests with $\Delta(r)$ at least 60 seconds. Requests with smaller $\Delta(r)$ include mostly requests for embedded content and are likely to utilize an existing persistent connection (Currently the default value used in the Apache HTTP server [2] leaves a connection open for 15 seconds past the last request arriving on it). Notice that since $\Delta(r)$ may change when simulating a single user versus when simulating all users the group of requests for which $\Delta(r)$ is greater than some threshold also changes. For the single user scenario we aggregated measurements for 52363 requests with valid HTTP response, and for the all users scenario we had 39651 requests. The number of different servers in both sets of requests was 13851. The choice of the simulation scenario barely affected the results both for search engine referrals and for the complete log so subsequently we show only the results simulating the scenario where all users are active. As in Section III we requested "home pages" of servers rather than the actual files since the latter's had a large fraction of obsolete names.

For each measurement of a particular server we timed DNS lookup (1), TCP connection-establishment (2), and HTTP `GET` request-response (3). We also timed a consecutive TCP connection-establishment (4) (warm-route), a consecutive HTTP `GET` request (5) (warm-server), and a consecutive HTTP `HEAD` request (6) (warm-server minimal transmission time).

B. Latency factors revisited

In this section we present more measurements of the various latency factors. In contrast with Section III we aggregate here on requests rather than on servers. By doing that we give higher weight to popular servers and take reference locality into account by associating a latency with a request r according to $\Delta(r)$.

Figures 4 and 5 show the relative contributions of the various latency factors for AltaVista referrals and for all request with $\Delta(r) \geq 60$, respectively. Smaller degree of locality of references to servers in the subsequence consisting of AltaVista referrals causes larger latencies. In both figures however, the cold-server effect (the gap between (3) and (5)) and the DNS lookup time are showing as significant contributors to latency and in particular to very long latencies. Even when aggregating across all requests DNS query time is contributing over 4 seconds for over 3% of the requests and HTTP RTT exceeds 1 second for about 5.5% of the requests and exceeds 4 seconds for over 2% of the requests. Transmission time (the gap between (5) and (6)) and warm route effect (gap between (2) and (4)) are less significant.

Connection-establishment time shows as a significant latency factor as well. Usually it is smaller than HTTP request-response as the latter incorporates transmission time and processing by the HTTP session. Figure 5, however, shows many more (warm) TCP connection-establishment times between 2 and 7 seconds (4) than the respective numbers of (warm) HTTP request-response times (5)-(6). In contrast with Figure 4 the plot in Figure 5 aggregates many measurements of relatively few popular servers (like search engines). Hence, the relative high frequency of large TCP connection-establishment times may indicate some additional processing or wait time of SYN segments at the TCP layer of relatively few popular servers. Different retransmission timeout values for the SYN segment and later segments may also be the reason for large connection-establishment times to congested servers (when the likelihood of loosing packets increases).

C. Simulation results

We calculated the perceived latency for each technique t by adding an appropriate subset S_t of latency factors. For each request r we added the items in S_t for the measurement that corresponds to $\Delta(r)$. The subset corresponding to each technique is as follows. (The numbers refer to the list of the six latency factors we measured at the beginning of this section.)

- with neither technique: (1)+(2)+(3).
- with pre-resolved DNS query: (2)+(3).
- with pre-resolved DNS query and a pre-warmed route: (3)+(4)
- with pre-connecting: (3)
- with pre-resolved DNS query, warm route, and warm server: (4)+(5)
- with pre-connecting and warm server: (5)

First we describe the results for search engine referrals. Figure 6 shows perceived latency with and without various techniques under the sufficient lead time assumption. We show Stanford-based measurements for requests referred by AltaVista. Results were consistent across the two groups of requests referred by AltaVista and Yahoo!, and the two locations, Stanford and AT&T Labs. Also, taking available lead time into account rather than assuming sufficient lead time had a small effect on the results. This indicates that most often the available lead time is sufficient for all pre-transfer techniques. Indeed, the lead time is between 3 seconds and 100 seconds for about 75% of the AltaVista and Yahoo referrals.

Perceived latency exceeds 1 second on about 22% of requests and exceeds 5 seconds for about 8% of requests. The respective percentages decrease to 10% and 3% with pre-resolving alone, 6% and 2.5% with pre-connecting and 3% and 1% with pre-warming (in addition to pre-connecting). Follow-Up requests after search engines queries have high fraction of requests to servers that were not accessed recently. Hence, cold servers and long DNS query times are prevalent and pre-resolving and pre-warming are particularly effective.

Figure 7 shows perceived latency on all the requests in the log (with $\Delta(r)$ greater than 60 seconds) with and without the various techniques. As in Figure 6 the line that corresponds to pre-resolving together with warming the route was almost identical to the line that corresponds to pre-resolving alone and therefore

[3] A secondary reason for considering this group of requests was our ability to obtain reasonably close estimates on lead time for them.

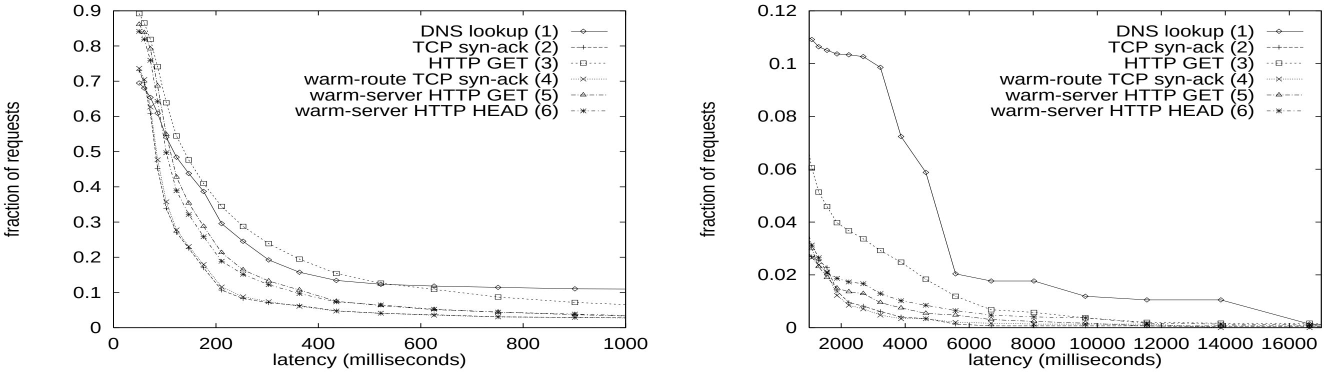

Fig. 4. Perceived latency and contribution of various factors on requests referred by AltaVista. Measured from Stanford.

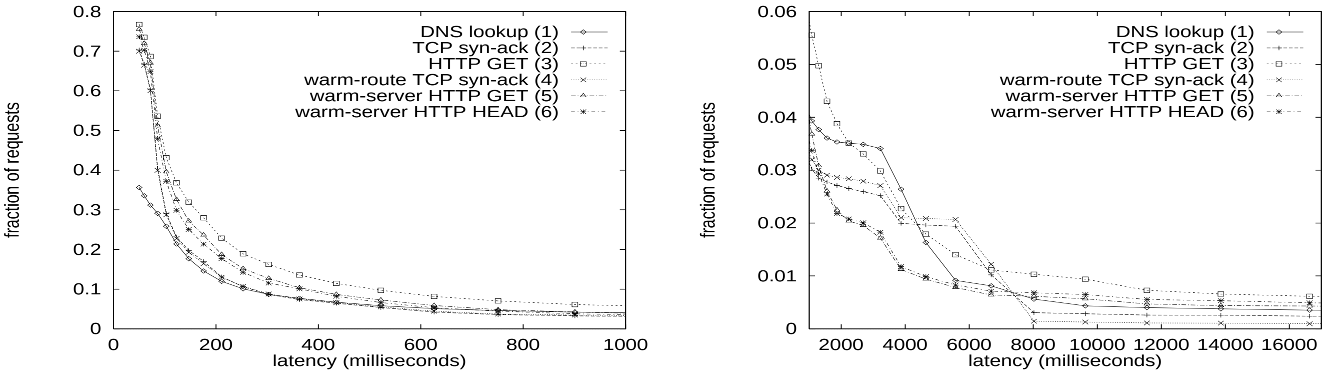

Fig. 5. Different factors of the perceived latency for all requests. Measured from AT&T labs.

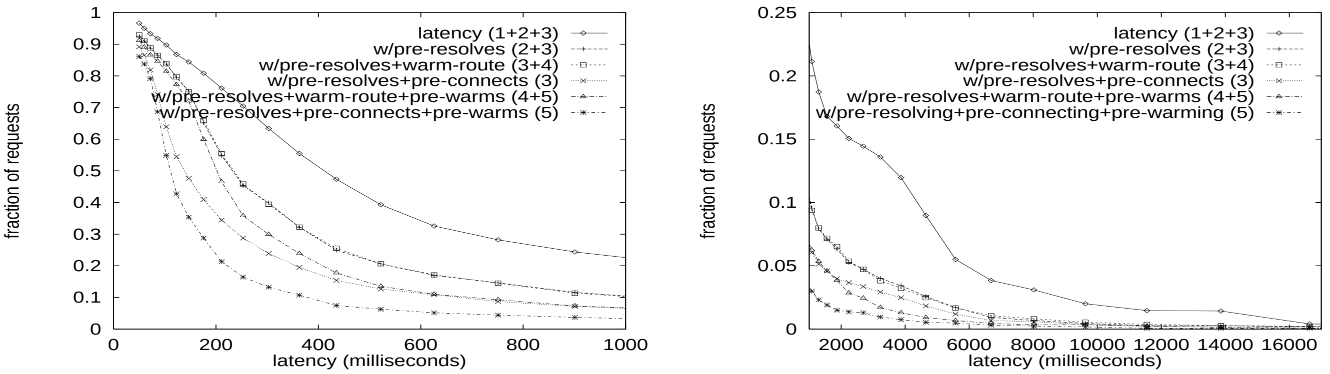

Fig. 6. Perceived latency with various prefetching techniques for AltaVista referrals, assuming sufficient lead time.

omitted. Perceived-latency exceeds 1 second for about 14% of requests and exceeds 4 seconds for about 7% of the requests. The respective percentages decrease to 10% and 4.5% with pre-resolving alone, 6% and 2% with pre-connecting and 4% and 1% with pre-warming (in addition to pre-connecting). The latency incurred on requests with $\Delta(r) < 60$ (typical requests for embedded objects) is reflected by (5) which assumes that all 3 pre-transfer techniques were performed. If pipelining or multiple connections are used for fetching embedded contents, then the network RTT cost is incurred once per multiple requests. In all, the frequency of longer wait times is significantly reduced when pre-resolving, pre-connecting, and pre-warming are applied.

VII. Future Research

Perhaps the biggest challenge for future work is to design a good implementation of pre-transfer techniques, at either browsers or proxies, that maximizes benefits while controlling overhead. We outline some further directions.

Prefetched idle TCP connections can be further used for cache validations, uncovering obsolete links, prefetching meta-data, and following redirected links. These operations have a positive side effect of "warming" the server. Prefetched meta-data can be used to annotate hyperlinks to assist users. Pre-connecting can also be used to select a mirror site (e.g., the one with the shortest RTT, or using some bandwidth-estimation.) The spirit of our techniques applies also when cooperative caching is deployed: Latency can then stem from identifying the source of data and establishing a connection to that source. This can be performed prior to the user's request. The effectiveness and the

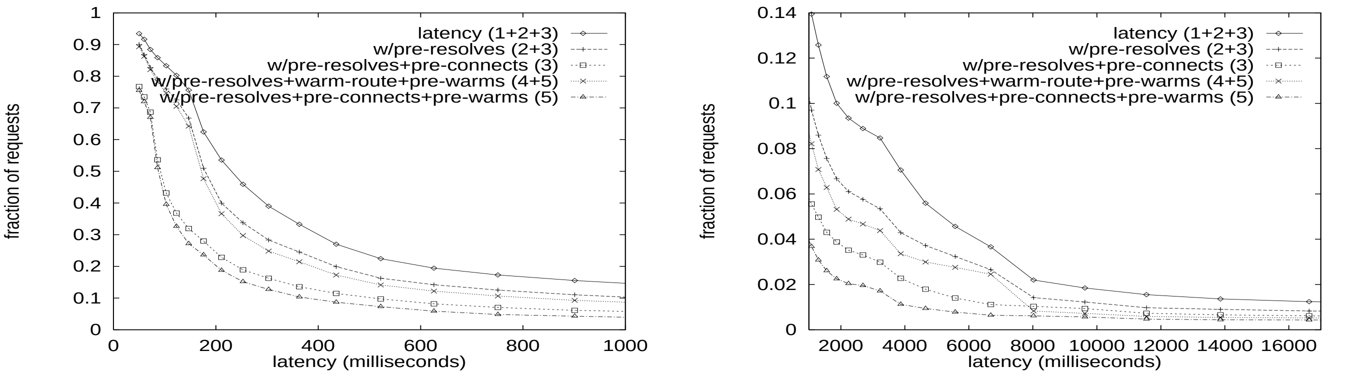

Fig. 7. Perceived latency with various prefetching techniques for all requests.

cost of each of these ideas is a topic for future research.

The discrepancy between the latency incurred on requests made to search-engines and latency on following referral requests can potentially be addressed at the search-engines sites. For example, by (1) annotating links with "pre-resolved" buttons ; (2) piggybacking resolutions along with the response page (e.g., using trailers as in [9]); (3) embedding invisible references to "href"ed servers that would cause the user's browser to pre-connect while processing the HTML response page; (3) act as proxies, fetching contents on behalf of the user while using pre-transfer techniques and sharing benefits across clients. This may also be combined with search-engines caching to eliminate staleness, and can also provide privacy to the user and valuable usage data for the search-engine site.

ACKNOWLEDGMENTS

The pre-connecting technique was conceived in a joint discussion with Uri Zwick. We thank Jeff Mogul and Rob Calderbank for their comments on an early version of this manuscript. We are grateful to Menashe Cohen and Yishay Mansour for many good suggestions and pointers. We also thank Yehuda Afek, Ramon Caceres, Dan Duchamp, Sam Dworetsky, David Johnson, Balachander Krishnamurthy, Ben Lee, and Jennifer Rexford, for interesting discussions on our ideas and their implications.

REFERENCES

[1] AltaVista. `http://www.altavista.com`.
[2] Apache HTTP server project. `http://www.apache.org`.
[3] G. Banga and P. Druschel. Measuring the capacity of a Web server. In *Proceedings of the USENIX Symposium on Operating on Internet Technologies and Systems*. USENIX Association, 1997. `http://www.cs.rice.edu/~gaurav/papers/web-paper.ps`.
[4] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (ODSI)*. USENIX Association, 1999. `http://www.cs.rice.edu/~gaurav/papers/osdi99.ps`.
[5] G. Banga and J. Mogul. Scalable kernel performance for Internet servers under realistic loads. In *Proceedings of the USENIX Annual Technical Conference*. USENIX Association, 1998. `http://www.cs.rice.edu/~gaurav/papers/usenix98.ps`.
[6] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. RFC 1945, MIT/LCS, May 1996. `http://ds.internic.net/rfc/rfc1945.txt`.
[7] E. Cohen and H. Kaplan. Proactive caching of DNS records: approaches and algorithms. Submitted, 1999.
[8] E. Cohen, H. Kaplan, and J. D. Oldham. Policies for managing TCP connections under persistent HTTP. In *Proceedings of the World Wide Web-8 Conference*, 1999.
[9] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the Web using server volumes and proxy filters. In *Proceedings of the ACM SIGCOMM'98 Conference*, September 1998.
[10] Duchamp. D. Prefetching hyperlinks. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1999. `http://www.usenix.org/events/usits99`.
[11] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: a live study of the world wide web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997. `http://www.usenix.org/events/usits97`.
[12] T. Faber, J. Touch, and W. Yue. The TIME_WAIT state in TCP and its effect on busy servers. In *Proceedings of the IEEE INFOCOM'99 Conference*, pages 1573–1583, 1999.
[13] A. Feldmann, R. Cáceres, F. Douglis, G. Glass, and M. Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *Proceedings of the IEEE INFOCOM'99 Conference*, 1999.
[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, and T. Leach, P. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2616, ISI, June 1999. `ftp://ftp.isi.edu/in-notes/rfc2068.txt`.
[15] H. Frystyk Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley. Network performance effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of the ACM SIGCOMM'97 Conference*, Cannes, France, August 1997.
[16] J. Heidemann, K. Obraczka, and J. Touch. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM transactions on networking*, 5:616–630, 1997.
[17] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM'88 Conference*, August 1988.
[18] B. Janssen, H. Frystyk, and Spreitzer M. HTTP-NG architectural model, August 1998. work in progress.
[19] Balachander Krishnamurthy and Craig E. Wills. Study of piggyback cache validation for proxy caches in the world wide web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997. `http://www.usenix.org/events/usits97`.
[20] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 13–22, December 1997.
[21] W. LeFebre and K. Craig. Rapid reverse DNS lookup for Web servers. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1999.
[22] J. C. Mogul. The case for persistent-connection HTTP. *Computer Communication Review*, 25(4):299–313, October 1995.
[23] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of the ACM SIGCOMM'97 Conference*, Cannes, France, August 1997.
[24] The Mozilla organization. `http://www.mozilla.org`.
[25] NCSA HTTPd. `http://hoohoo.ncsa.uiuc.edu/`.
[26] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. In *Proceedings of the ACM SIGCOMM'98 Conference*, 1998.
[27] S. Sen, , J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proceedings of the IEEE INFOCOM'99 Conference*, 1999.
[28] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, Reading, MA, 1994.
[29] Z. Wang and P. Cao. Persistent connection behavior of popular browsers.
[30] Yahoo! `http://www.yahoo.com`.